Search for mixing of D^0 and its antiparticle using neural networks

A Senior Honors Thesis

Presented in Partial Fulfillment of the Requirements for graduation
with distinction in Physics in the undergraduate colleges
of The Ohio State University

by

Austen Rau

The Ohio State University
July 2006

Project Advisor: Professor Richard Kass, Department of Physics

Abstract

Mixing is a process in which a particle spontaneously turns into its antiparticle. The Standard Model of particle physics at the box diagram level predicts that mixing of a D^0 should occur approximately once every ten billion decays, while other theories predict much larger mixing rates [1,2]. Measurement of the mixing rate of the D^0 is an important test of the Standard Model. If mixing rates are larger than what the Standard Model predicts, this could be evidence of physics beyond the Standard Model and would be a major physics discovery. Since the D^0 has zero electric charge and a lifetime of only 4×10^{-13} seconds, limitations in current elementary particle detector technology require examination of the decay products of the D^0. Some other processes have decay products that are similar to the decay products resulting from the mixing of a D^0, so it is necessary to distinguish these processes. In this analysis neural networks are used to help determine whether a D^0 decay involved mixing or another process.

Neural networks are models of complicated functions that, given a number of inputs, will attempt to predict the value of one or more outputs. In this study, the inputs consist of observables measured by the BaBar detector at the Stanford Linear Accelerator Center [3]. The neural network output is binary, with 1 representing the signal candidates and 0 representing all background events. The signal mode is a D^0 decaying into K^+, an electron, and an antineutrino, while the D^0 normally decays into K^-, a positron, and a neutrino. In this study we use the neural network computer program MLPfit [4]. MLPfit requires training data which is used to develop the neural network. This training data consists of inputs that are known to produce a certain output. These neural networks were trained with MLPfit using Monte Carlo simulation data from the BaBar experiment. The goal is to produce a neural network that will provide a cleaner data sample with fewer background events than the neural network that is currently being used in

the experiment. Comparisons show neural networks trained by MLPfit are comparable to the neural network that is currently being used in the experiment.

Introduction

Mixing is a process that occurs when a particle spontaneously turns into its antiparticle. Mixing has already been detected for K^0 and B^0 but not D^0, which are all bound states of a quark and an antiquark ($K^0 = d\bar{s}$, $B^0 = d\bar{b}$, $D^0 = c\bar{u}$) [5]. The Standard Model of particle physics at the box diagram level predicts a mixing rate of approximately 1 in every 10 billion events. Another theory within the Standard Model, the dispersive theory, predicts larger mixing rates than the theory represented by the Feynman box diagram. Figure 1 shows the Feynman box diagram for the mixing of D^0, with W acting as the mediator of the weak force. If mixing rates are larger than what the Standard Model predicts, this could be evidence of physics beyond the Standard Model because it would imply that a mediator that is heavier than the W or quark heavier than the b are involved in this process. This analysis will test the Standard Model.

The lifetime of a D^0 is only 4×10^{-13} seconds and it has zero electric charge, which make it very difficult to detect a D^0. So we are forced to examine the decay products to detect mixing. Processes with similar decay products can mimic mixing, so we must use some method of data analysis to determine which processes are mixing and which are background.

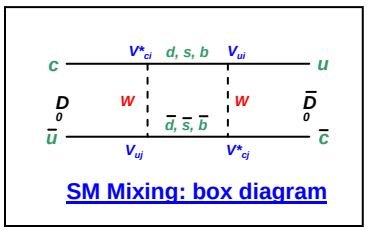

Figure 1-The Feynman box diagram representing D^0 mixing.

Neural networks are models of complex functions which can be used for such an analysis. The neural networks used in this project were trained using MLPfit [4], a neural network software package designed at the European Laboratory for Particle Physics (CERN). The inputs to the neural networks consist of various observables measured by the BaBar detector at the Stanford Linear Accelerator Center (SLAC) [3]. The neural networks then produce a binary output, with an output of 1 representing a signal event and 0 representing some other process. This analysis was intended to reduce the number of background events in a data set so that the mixing rate of D^0 could be measured. The neural networks trained by MLPfit currently produce comparable results to the neural network that is currently being used in this analysis.

Mixing of D^0

Mixing is a process that occurs when a particle spontaneously turns into its antiparticle. This process occurs by the weak interaction, which does not conserve charm. This is evident in the mixing of D^0 because it contains a charm quark and thus a charm of +1 and its antiparticle has a charm of -1 because it has an anticharm quark. For charm to be conserved, the value must be the same before and after the process. Figure 2 shows the normal decay of D^0 and the decay of D^0 via mixing.

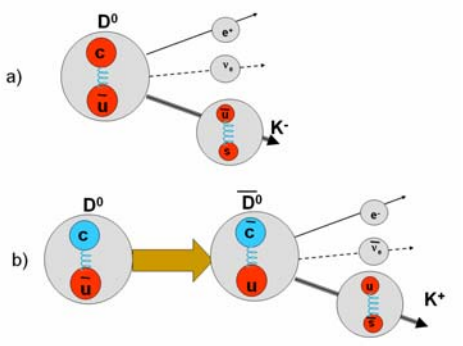

Figure 2-a) Decay of *D*0. b) Decay of *D*0 via mixing

For mixing to occur, this process must obey certain conservation laws. Since there are no other decay products in this process, the additive quantum numbers (electric charge, baryon number, and lepton number) must equal zero. Mixing has already been detected for K^0 and B^0 but not D^0, which are all bound states of a quark and an antiquark. Since quarks have baryon number +1/3 and antiquarks have baryon number -1/3, the total baryon number for these particles is zero. The lepton number for these particles is also zero, because there are no leptons involved in this process. Total electric charge is also zero for these particles ($q_d = -\frac{1}{3}$, $q_{\bar{s}} = +\frac{1}{3}$, $q_{\bar{b}} = +\frac{1}{3}$, $q_c = +\frac{2}{3}$, $q_{\bar{u}} = -\frac{2}{3}$, in units of proton charge) [5]. Because the additive quantum numbers equal zero, they will all be conserved if mixing occurs.

D^0 and its antiparticle are created in particle accelerators by colliding electrons and positrons together. The charm quarks in the D^0 are created via the electromagnetic interaction and are then bound to other quarks via the strong interaction. Therefore $\left|D^0\right\rangle$ and $\left|\overline{D}^0\right\rangle$ are eigenstates of the strong Hamiltonian. These eigenstates can also be written as superpositions of $\left|D_1\right\rangle$ and $\left|D_2\right\rangle$, which are eigenstates of CP and (neglecting CP violation) the weak interaction. Thus $\left|D^0\right\rangle$ and $\left|\overline{D}^0\right\rangle$ can be written as the following[1]:

$$\left|D^0\right\rangle = \frac{1}{\sqrt{2}}\left(\left|D_1\right\rangle + \left|D_2\right\rangle\right) \qquad \left|\overline{D}^0\right\rangle = -\frac{1}{\sqrt{2}}\left(\left|D_1\right\rangle - \left|D_2\right\rangle\right)$$

As the particles propagate in a vacuum, however, they no longer remain in eigenstates of the strong Hamiltonian. As the postulates of quantum mechanics state, the states evolve in time according to the time-dependent Schrödinger equation:

[1] The following calculation was adapted from a similar calculation for the K^0 [6].

$$i\hbar\frac{\partial|\psi(t)\rangle}{\partial t} = H_{eff}|\psi(t)\rangle$$

The eigenstates of the effective Hamiltonian have eigenvalues $m_1 - \frac{i}{2}\gamma_1$ and $m_2 - \frac{i}{2}\gamma_2$ in the rest frame of the particle, where m_j and γ_j represent the mass and decay width of D_j. D_1 and D_2 have slightly different masses and lifetimes because they have different eigenvalues of the CP operator. Applying these eigenvalues to the Schrödinger equation, the corresponding time-dependent states are:

$$|D_1(t)\rangle = e^{-(i/\hbar)(m_1-(i/2)\gamma_1)t}|D_1\rangle \qquad |D_2(t)\rangle = e^{-(i/\hbar)(m_2-(i/2)\gamma_2}|D_2\rangle$$

The factor $e^{-(im_j/\hbar)t}$ provides the oscillating behavior of the wavefunction with $E_j = \sqrt{p_j{}^2 + m_j{}^2} = \sqrt{m_j{}^2} = m_j$, while $e^{-(\gamma_j/2\hbar)t}$ accounts for the exponential decay of the particle with lifetime $\tau_j = \frac{\hbar}{\gamma_j}$. These time-dependent states can be used to calculate the probability of finding a $\overline{D}^0$ in a pure beam of D^0 at a later time:

$$P(\overline{D}^0,t) = \left|\langle\overline{D}^0|D^0(t)\rangle\right|^2 = \frac{1}{4}\left|\left(e^{-(i/\hbar)(m_1-(i/2)\gamma_1)t} - e^{-(i/\hbar)(m_2-(i/2)\gamma_2)t}\right)\right|^2$$

$$= \frac{1}{4}e^{-t/\tau_1} + \frac{1}{4}e^{-t/\tau_2} - \frac{1}{2}e^{-(1/2)(1/\tau_1+1/\tau_2)t}\cos\frac{\Delta mt}{\hbar}$$

By examining the decay in terms of the eigenstates of the strong and weak interactions, it is clear that we can obtain a $\overline{D}^0$ in a beam of pure D^0, which occurs through the process of mixing.

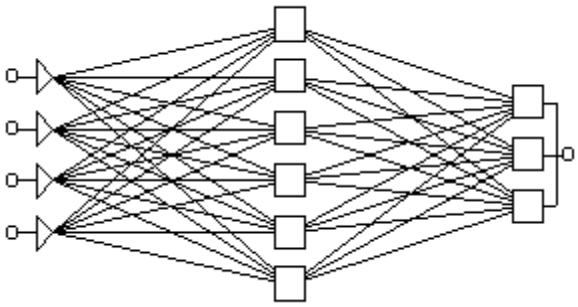

Figure 3-An example of the structure of a neural network with 4 inputs, 6 hidden neurons, and 3 outputs.

Neural Networks

Neural networks are a means of modeling complicated functions. They are designed as an extremely simplified model of the human brain with feedforward structure. They consist of usually three layers of "neurons" connected by "synapses." Figure 3 shows an example of a neural network structure with 4 inputs, 6 neurons in the hidden layer, and 3 outputs. A neuron represents a value, while a synapse represents a weight. Each neuron is simply a weighted sum of the neurons of the previous layer. The layers in a neural network are the input, hidden, and output layers. Between each of these layers is a transfer function which each neuron is passed through before summation. The transfer function between the input and hidden layers is linear, while between the hidden and output layers it is a sigmoid function, defined as $y = \frac{1}{1+e^{-x}}$. The sigmoid function serves two purposes. A linear combination of sigmoid functions can approximate any continuous function, and they also take values in a wide range and produce a value in a much smaller range. This allows the neural networks to approximate complicated functions.

MLPfit is a neural network software package written in C++ that is used to train neural networks. This program requires two sets of data, the learning data and the testing data, with

known inputs and outputs. The number of inputs, outputs, and neurons in the hidden layer as well as the number of iterations must be defined before training. The neural network begins with weights which can either be values specified by the user prior to training or random values between -1 and 1. The inputs from the learning data are fed into the neural network to produce an output. This neural network output is compared to the known output from the data set. An error function between the neural network output and the known output is used to determine the effectiveness of the neural network and is defined as $E = \frac{1}{2}\sum_p \omega_p (o_p - t_p)^2$, where ω_p is a weight for each event that typically equals 1. The weights in the neural network are changed iteratively to minimize this error. Each iteration is called an epoch. The error between the testing data and the neural network output must also converge to ensure that the neural network is also effective on data that is not used in the training process.

MLPfit provides many different methods of error minimization. The method used in this project is the Broyden, Fletcher, Goldfarb, and Shanno (BFGS) method [7, 8, 9, 10], named for its creators. This method falls into the category of methods with line search, which has four steps. The error is treated as a function of the vector $\vec{w}_t$, which contains all of the weight values. The direction of steepest decent $\vec{s}_t$ is calculated from the gradient of the error function. The next step, called the line search, searches for a coefficient α which minimizes the value $E(\vec{w}_t + \alpha\vec{s}_t)$. The vector containing all of the weights is then updated ($\vec{w}_{t+1} = \vec{w}_t + \alpha\vec{s}_t$). This process is then repeated until the error is minimized.

MLPfit displays the error value for both the learning and testing data during each iteration. This allows the user to ensure that the error function is actually decreasing and converging. After the neural network training, MLPfit creates a neural network function written

in either C or FORTRAN with the weights equal to the values given after the final iteration, as well as a file containing all of the final weight values. This function must then be tested before it can be used in analysis.

Early Studies—Two Gaussian Distributions

At the beginning of this project, it was necessary to become familiar with MLPfit. This was done by examining the output of a neural network trained using generated data consisting of two Gaussian distributions. These distributions were created by adding twelve random numbers between 0 and 1, which approximates a Gaussian distribution with a mean value of 6 and variance of 1 according to the Central Limit Theorem [11]. By adding a constant value to half of the events, the data was separated into two distinct Gaussian distributions with the same number of events but different mean values. Each distribution was then assigned an output value. Events that fell into the leftmost distribution would correspond to an output of 1 (signal), while events in the rightmost distributions would have an output of 0 (background). This binary output is typical of classification problems. These studies were used to determine the effect of certain variables, such as number of events, amount of separation, and number of neurons in the hidden layer, on the neural network output.

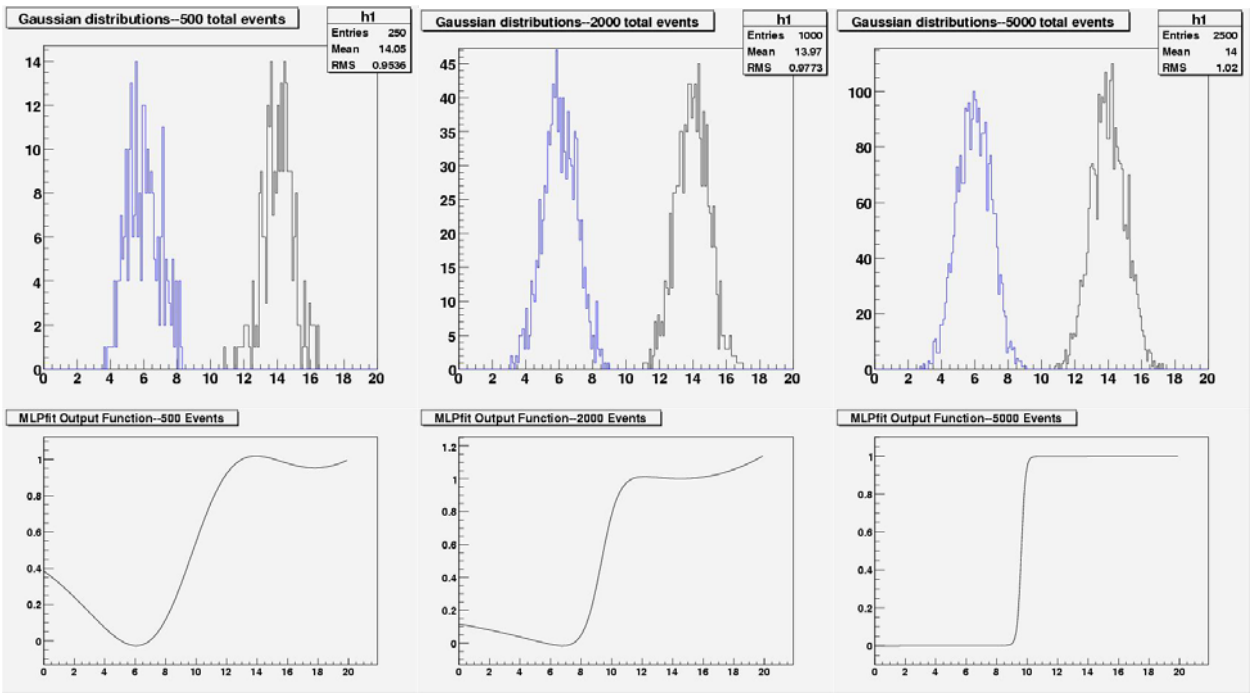

Figure 4-The effect of data set size on the neural network output. As the number of data points increase, the output more closely resembles a theta function.

The number of events used in neural network training strongly affects the output of the neural network. A larger data set provides MLPfit with more information about the data, which allows it to better approximate a function. By training neural networks with data sets of varying size, importance of using large data sets in neural network training can be shown. In the case of two non-overlapping Gaussian distributions with mean values of 6 and 14 produced as described above, an infinitely large data set would produce a theta function as the output function, with all events falling in the leftmost distribution yielding an output of 0 and all events contained within the rightmost distribution giving an output of 1. However, with small enough data sets, the output appears much different than a theta function. In this study Gaussian distributions with 500, 2000, and 5000 total events were used in neural network training. The corresponding output functions become more like a theta function with increasing data size, which can be seen in Figure 4.

Another important factor in the output of neural network training is the amount of separation between the distributions of the input variables. The effect of separation on the neural network output can be examined by changing the mean value of the rightmost distribution such that it begins to overlap with the other distribution. As we have seen, a neural network trained with non-overlapping distributions will produce an output that closely resembles a theta function. But as the distributions begin to overlap, the output of the neural network becomes less definite, falling somewhere between 0 and 1 instead of giving exactly 0 or 1. This output behavior occurs because, when the distributions overlap, the neural network has much more difficulty determining signal from background. Although the neural network cannot give a definite distribution to which a given event belongs, the likelihood that the event belongs to a distribution can be determined from the output. Events with outputs less than 0.5 are more likely to be background, while events with outputs greater than 0.5 are more likely to be signal.

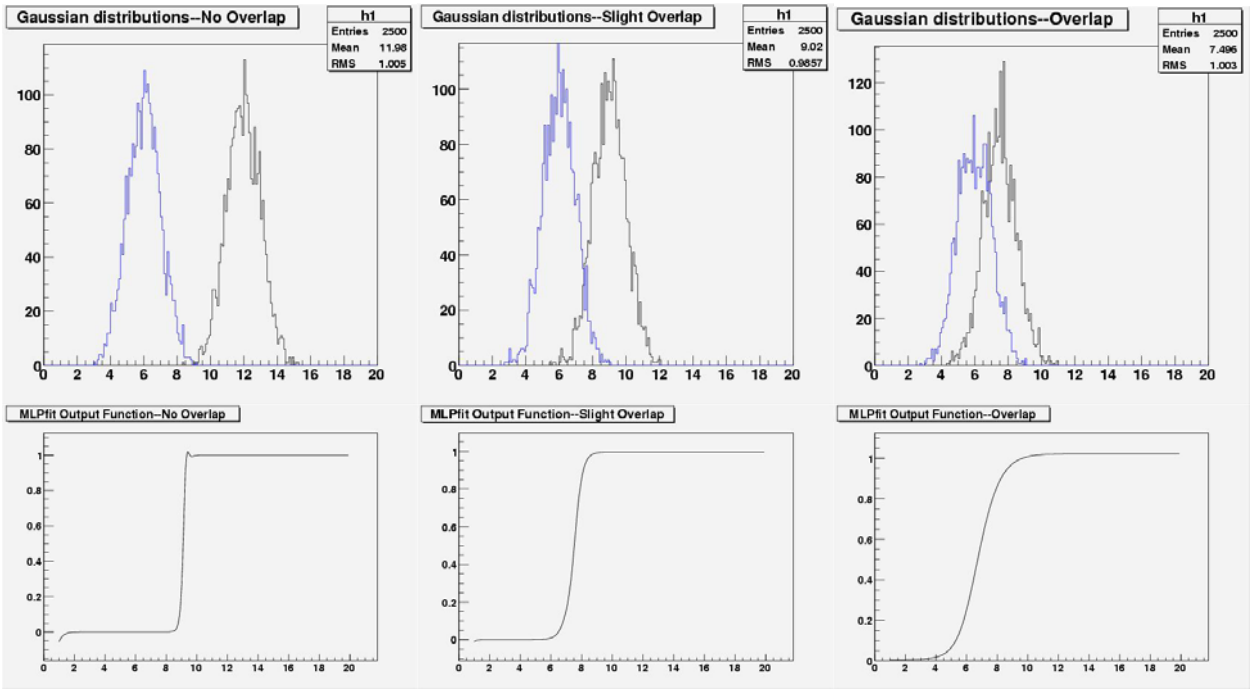

Figure 5-The effect of separation on the neural network output. Better separation yields an output function that more closely resembles a theta function.

Figure 5 shows a study with three different degrees of overlap. The first plot shows two distributions with mean values of 6 and 12 which are almost completely separated. The range within which the output is not nearly 0 or 1 is quite small, and this output function is a good approximation of a theta function. The second plot shows two distributions with mean values of 6 and 9. These distributions also overlap in this range, which is precisely the range in which the output falls between 0 and 1. The final plot shows distributions with mean values of 6 and 7.5 which overlap between 4 and 9. The same behavior can be seen once again. From these plots it is clear that a strong correlation exists between the range of overlap and the output falling between the expected values, which demonstrates the importance of using a data set with good separation in neural network training.

In addition to the amount of separation and size of the data set, the number of neurons in the hidden layer strongly affects the output of the neural network. A neural network with too few hidden neurons will be too simple to produce an accurate output, whereas too many hidden neurons could result in overfitting, creating a much more complex function than is necessary to produce an accurate output. As with the other variables, it is necessary to train neural networks with different numbers of hidden neurons in order to train the most effective neural network, although a good starting point is set the number of hidden neurons equal to half of the sum of the number of inputs and outputs [12].

Figure 6 shows two plots trained with the same data set but with a different number of neurons in the hidden layer. The first plot represents the output of a neural network with 3 neurons in the hidden layer. The output function strongly resembles a theta function, which implies that 3 hidden neurons is an appropriate amount for this particle problem. The second

plot represents the output function of a neural network with 6 hidden neurons. This function looks slightly less like a theta function than the previous plot because the range of uncertainty is about 1 output unit wider. Because of this behavior, this neural network structure is not the best function for this particular case.

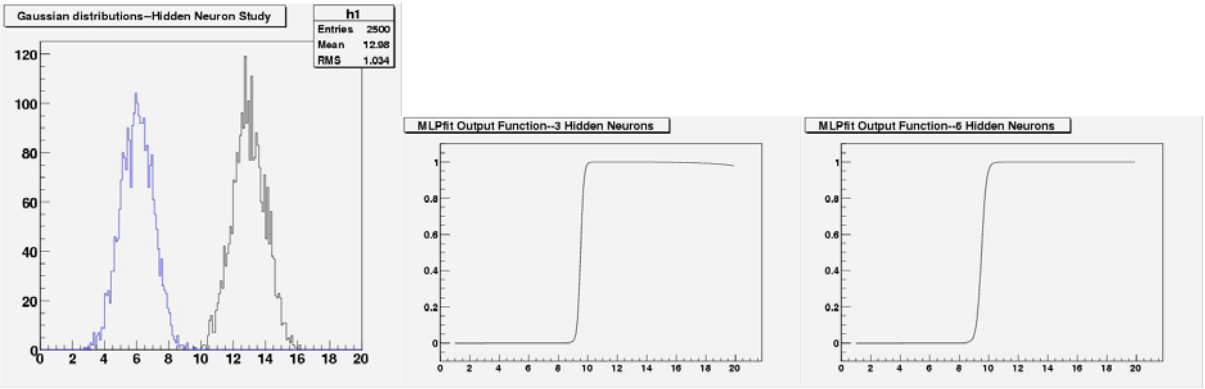

Figure 6-The effect of the number of hidden neurons on the neural network output. Too many hidden neurons can reduce the performance of the output function.

Neural Network Study with BaBar Data

After familiarization with MLPfit through the trivial case of two Gaussian distributions, neural networks could be applied to the task of detecting the mixing of D^0. In this study neural networks were trained using Monte Carlo data from the BaBar experiment. Because the Monte Carlo data is a simulation of real data from the BaBar detector, there are a vast number of available observables that could potentially be useful inputs. The distributions of the potential input variables must be examined to determine whether a given input has enough separation to improve the neural network output. Once the useful inputs are found, neural networks with different combinations of these variables must be trained. The output of the MLPfit neural networks must then be compared to the output of the neural network that is currently being used in the experiment to determine how the MLPfit output compares to the current neural network

output. These comparisons can be made by using values such as $\frac{signal}{\sqrt{background}}$ and efficiency, which is defined as the signal retained divided by the total signal. The signal over the square root of background is a measure of the number of signal events that would be kept if a cut were to be made at a given value relative to the number of background event. A value of -1 is returned if there are no background events. These values will help to determine which neural network better separates signal from background.

This data differs from the data used to train the neural network that is currently being used in the experiment because it is double-tagged instead of only single-tagged. Single-tagged data uses information about the slow pion from the D^* decay to eliminate background, whereas double-tagged data examines both the slow pion and the decay products of the recoil D meson. The five tagging modes that were selected are a D^0 decaying into a kaon and a pion, a D^+ decaying into a kaon and two pions, or a D^0 decaying from a parent D^* into a kaon and a pion, a kaon and two pions, or a kaon and three pions. These tagging modes were selected both because they occur more frequently than other similar decays and because they are easy to reconstruct since nearly all of the decay products have net electric charge.

The signal mode for this data is both a D^{*+} decaying into a π^+ and a D^0, which then decays via mixing into a K^+, an electron, and an antineutrino, or a D^{*-} decaying into the charge conjugate of this decay. The background modes include a D^0 signal candidate combined with an uncorrelated slow pion candidate, so the decay products appear the same as in the signal mode. The other background modes are due to a D^0 decay reconstructed with a correct slow pion and kaon candidate but a background electron candidate, or a correct electron and a background kaon candidate. There is also a background mode consisting of all uncorrelated components, but this effect is small. These semi-leptonic modes were selected for this analysis because hadronic

decays of D^0 via mixing yield the same decay products as a different process, and it is difficult to distinguish the process involved in the decay.

Examination of the distributions of many potential input variables led to the selection of the five variables whose distributions are shown in Figure 7. These variables represent (in the center of mass frame) both the transverse and longitudinal components of the momentum of the slow pion with respect to the momentum of the recoil *D* meson, the angle between the kaon-electron system and the slow pion, the angle between the slow pion and the recoil *D* meson, and the angle between the event thrust and the slow pion. The sixth variable is an example of a variable that does not have good separation and would not be useful in neural network training. The five useful variables were then used to train a neural network using MLPfit. The neural network has four neurons in the hidden layer and was trained for 400 epochs. The learning data consisted of 831912 events and the test data file contained 277181 events.

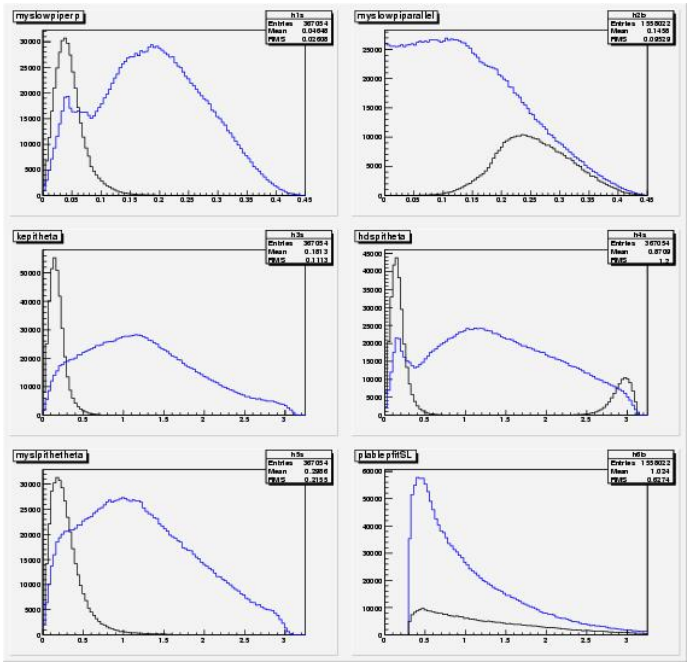

Figure 7-Distributions of the input variables for first study.

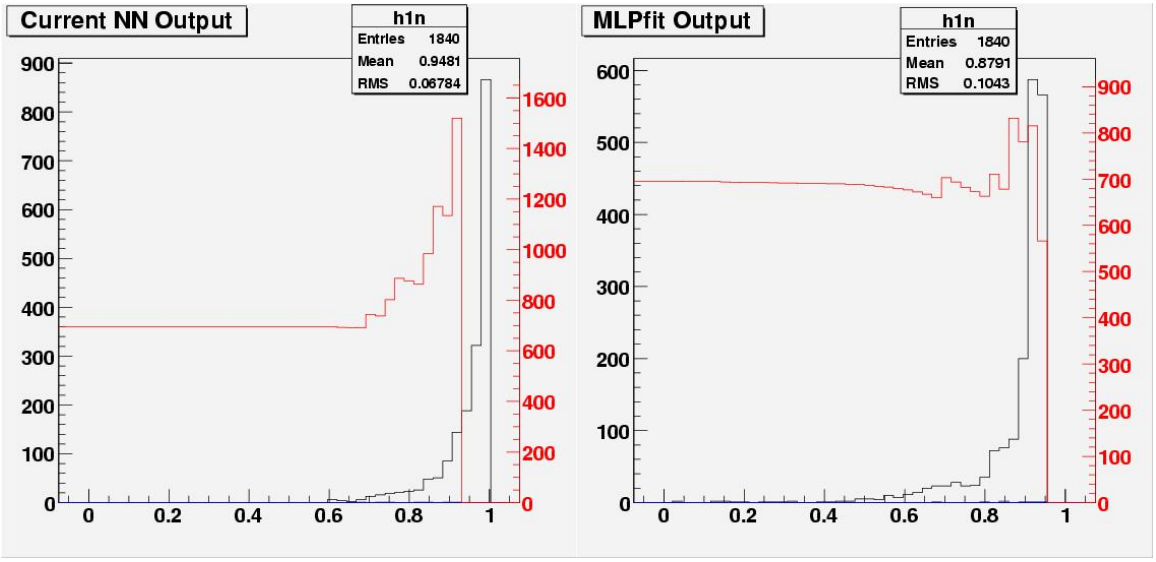

Figure 8-Output functions of the current neural network and MLPfit neural network with signal over square root of background as a figure of merit.

Figure 8 shows the output plots of both neural networks after cuts were made. In this particular study the current neural network performs much better than the MLPfit neural network. Although the signal distributions have similar shapes, the MLPfit neural network allows many more background events with outputs near 1, whereas the current neural network allows no background events with output greater than 0.9375. The signal over the square root of background for the current neural network peaks near 1500, while it peaks at only 840 for the MLPfit neural network.

The MLPfit neural network also produces the odd behavior that the signal peaks at 0.95 instead of 1. This behavior is a result of the fact the signal distributions completely overlap with the background distributions. This can be proved using the trivial case of a Gaussian distribution falling completely within constant distribution, shown in Figure 9. If the Gaussian distributions is defined as signal and the constant distribution is defined as background, the output for the signal distribution peaks at approximately 0.85, while the output for the background distribution peaks at 0. By reversing the signal and background, the opposite behavior occurs, with the

signal output peaking at 1 and the background output peaking at 0. This is evidence that the neural network tends to shift the output of a distribution that always falls within the range of another distribution, because no events exist that the neural network can assign the correct output with absolute certainty.

The slow pion momentum components are variables that are used in the cuts. Because they are also used in the cuts, they cause a bias in the data if they are used as inputs to the neural network. These cuts are so effective that they should only be used as cuts instead of input variables. Therefore it is necessary to select different variables as inputs to the neural network.

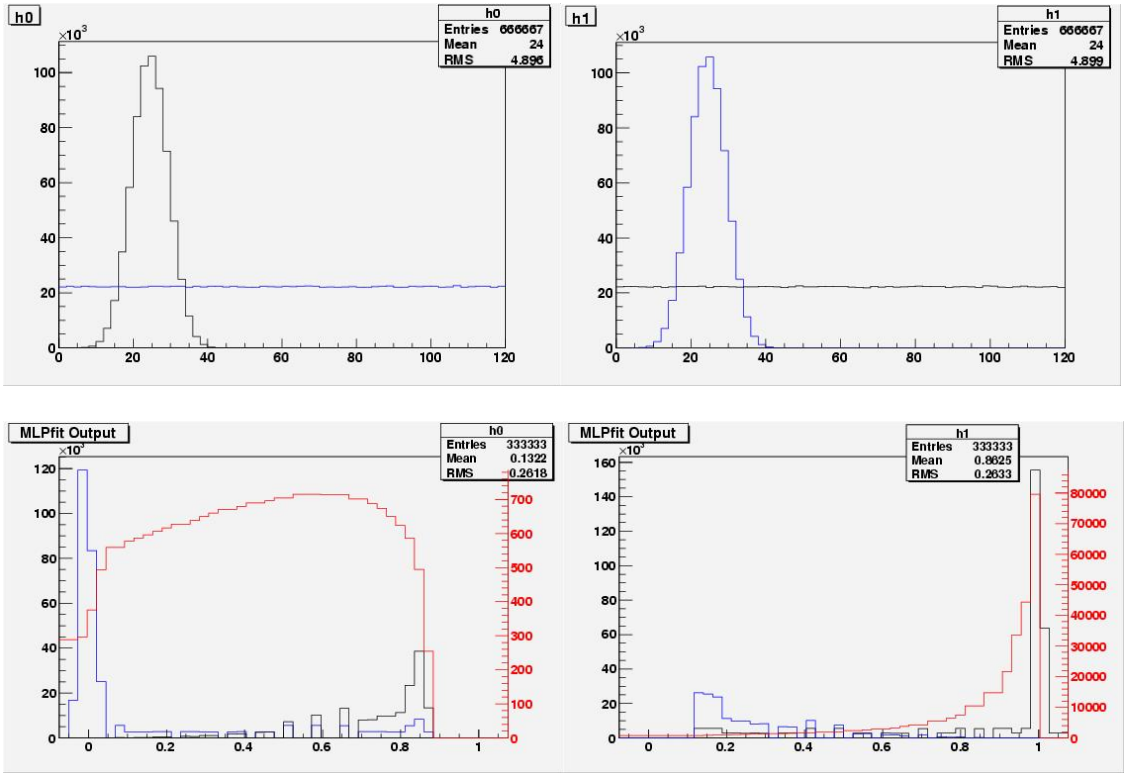

Figure 9-Displacement of the output peaks occurs when one distribution is completely overlapped by another.

Another study used some other variables to train the neural networks. These variables are all in the center of mass frame and include the momentum of the kaon-electron vertex, the opening angle between the thrust and the kaon-electron vertex, the opening angle between the kaon and electron, the opening angle between the tag pion and the thrust, and the opening angle between the tag pion and the kaon-electron vertex. The distributions of these variables can be seen in Figure 10.

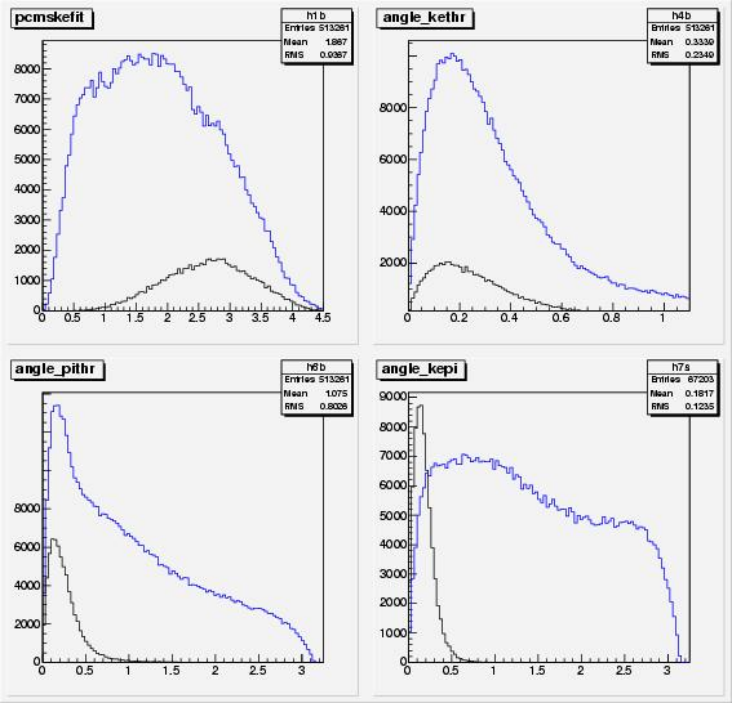

Figure 10-Distributions of input variables for final study.

The neural network in this study was trained over 400 epochs with 3 neurons in the hidden layer, 248211 events in the learn file, and 82674 events in the test file. Before any cuts are made, comparisons of the outputs in Figure 11 show that the MLPfit neural network performs much better than the current neural network because the latter retains almost as many background events near output 1 as signal events. Signal over the square root of background peaks at only about 105 for the current neural network, while it peaks at 280 for the MLPfit neural network. However, after the cuts the outputs are strikingly similar. Comparisons of the

outputs show that their performances are virtually identical if cuts are made at 0.5 on the MLPfit output and 0.9 on the current neural network output. The signal over the square root of background peaks at approximately 600 for the MLPfit neural network and at about 580 for the current neural network. Efficiency is also quite similar, with MLPfit producing about 85% efficiency and the current neural network retaining approximately 87% of the signal events. Both neural networks also retain slightly less than half of the background events.

Conclusion

In this analysis the neural networks trained with MLPfit are quite effective at separating signal from background. When comparing these neural networks with the neural networks that are currently being used in the BaBar experiment, it is clear that their performance is incredibly similar after making the cuts. Before the cuts, however, MLPfit is much better at producing a function that separates signal from background. We conclude that there has been no significant improvement in performance from the current neural network using these combinations of input variables and neural network structures, although a comparable neural network has been created.

Acknowledgements

I would like to thank both Professor Richard Kass and Dr. Amir Rahimi for continuously guiding me through this project and helping me to better understand high energy physics. I would especially like to thank Dr. Rahimi for repeatedly providing me with new Monte Carlo data to train the neural networks. I would also like to thank the College of the Arts and Sciences and the Ohio State University chapter of Sigma Xi for funding this project. Finally I would like to thank the BaBar experiment for giving me this opportunity to help find the mixing of D^0.

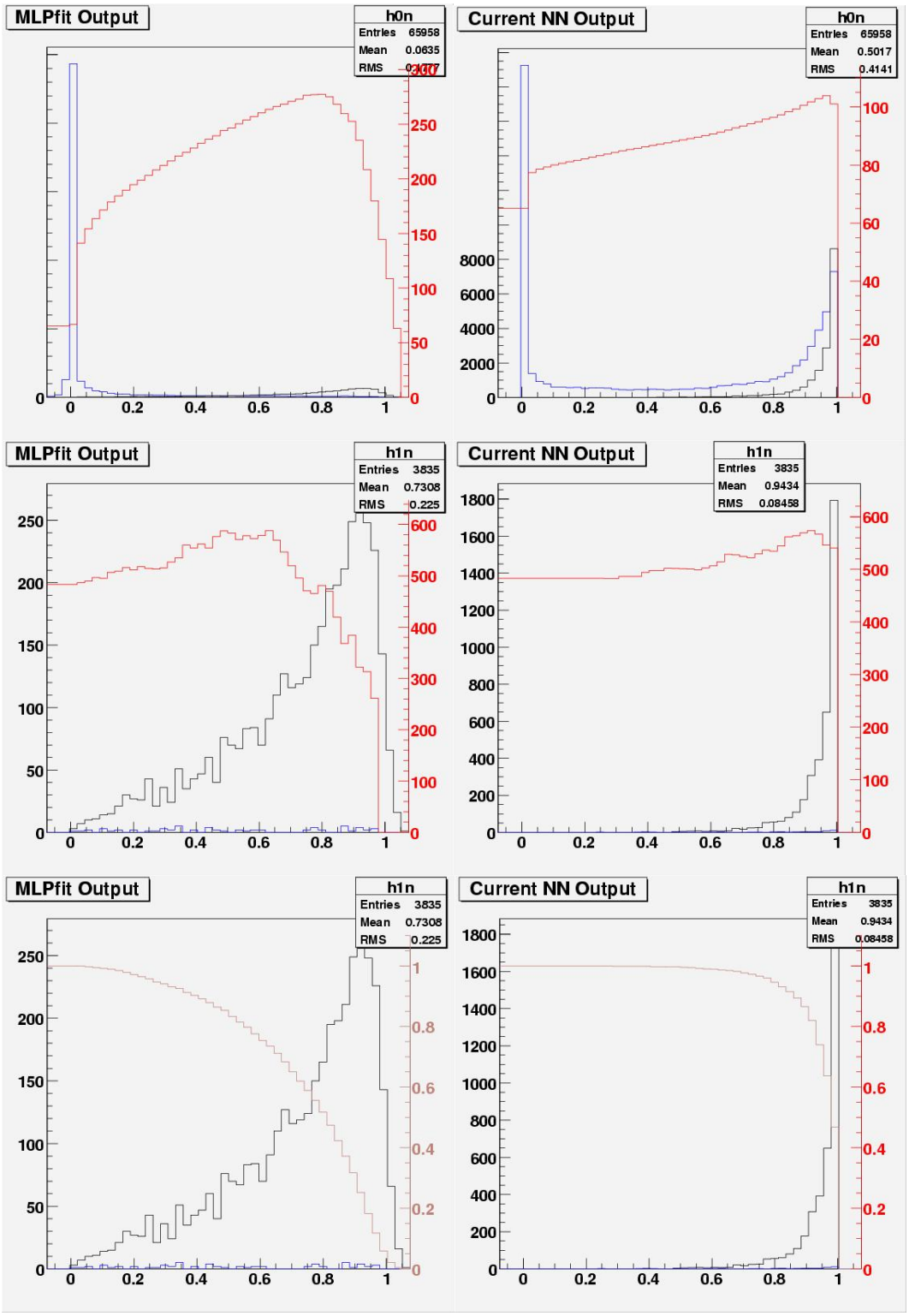

Figure 11-Output of the current and MLPfit neural networks. The first two graphs simply show the output functions of the neural networks with signal over square root of background as a figure of merit. The second two graphs show the output after the cuts were made with the same figure of merit. The third two graphs show the output function after the cuts while using the efficiency as a figure of merit.

References

1. P. Colangelo, G. Nardulli and N. Paver. On D0-D0bar mixing in Standard Model. *Phys. Lett.,* B242:71-76, 1990.
2. T. A. Keading. D-meson mixing in broken SU(3). *Phys. Lett.,* B357:151, 1995.
3. *BaBar Collaboration Home Page*. BaBar Collaboration. August 2005. http://www.slac.stanford.edu/BFROOT/
4. *MLPfit Home Page*. CERN. July 2006. http://schwind.home.cern.ch/schwind/MLPfit.html
5. Griffiths, David. *Introduction to Elementary Particles*. USA: John Wiley and Sons, Inc., 1987.
6. A. Das, T. Ferbel. *Introduction to Nuclear and Particle Physics*. USA: John Wiley and Sons, Inc., 1994.
7. C. G. Broyden. *Journal of the Institute of Mathematics and Its Applications* 1970, *6*, 76-90.
8. R. Fletcher. *Computer Journal* 1970, *13*, 317.
9. D. Goldfarb. *Mathematics of Computation* 1970, *24*, 23.
10. D. F. Shanno. *Mathematics of Computation* 1970, *24*, 647.
11. R. J. Larson, M. J. Marx. *An Introduction to Mathematical Statistics and its Applications*.
12. *Neural Networks*. StatSoft, Inc. August 2005. http://www.statsoft.com/textbook/stneunet.html